Continuous integration and delivery applied to large-scale software-intensive embedded systems

Martensson, Torvald

Chapter 1

Introduction

1.1 Towards Continuous Integration and Delivery

1.1.1 From Waterfall to Iterative Development

Software development has become one of the world's most important technologies. Software is today encapsulated in many of the things that we use in our everyday life – your car, your mobile phone or your camera. As software systems have grown and become more complex, the software industry has implemented various development methodologies – process frameworks we use to structure, manage and control our work.

The software industry emerged during the 1950s and 1960s, to a large extent using a "code and fix" approach. In the 1970s, the stage-gated waterfall model (Royce 1970) came into use as a tool to better predict and control large-scale software projects. Development is then conducted in separate phases to handle requirements, design, implementation, verification and deployment. The model appears to be extremely logical and prescriptive, but the problem is that in a large-scale and complex system it is close to impossible to specify all requirements and set the design prior to implementation.

As a response to this, iterative and incremental development models emerged during the 1980s and 1990s. The spiral model (Boehm 1988) is often mentioned as trendsetter for this new type of development models. The spiral models was followed by other iterative models such as Rapid Application Development (RAD) or Rational Unified Process (RUP).

1.1.2 Continuous Integration and Continuous Delivery

Starting in the late 1990s, a range of more adaptive development frameworks were proposed such as Scrum and XP. In parallel, continuous integration was introduced as a practice which could mitigate problems with a long and unpredictable integration process at the final stage of a project.

Continuous integration was introduced as a part of XP and other agile methodologies, but a clearly formulated definition of the practice is often traced to Martin Fowler: "a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day" (Fowler 2006). According to Fowler (2006), integration should be treated as a "non-event", instead of as a rare and therefore unpredictable process.

The value of continuous integration is according to Duvall (2007) to *reduce risks*, *reduce repetitive manual processes*, *generate deployable software*, *enable better project visibility*, and *establish greater confidence*. Continuous integration reduces risks by rebuilding and testing the software frequently, which reduces assumptions

about the software's behaviors. As the build and test process is automated, this reduces repetitive tasks, freeing people to higher-value work. As defects are detected and fixed sooner, the product can be released and deployed at any point in time instead of going through a long and unpredictable release process. Continuous integration also enables better project visibility, as it provides just-in-time information on the recent build status and quality metrics. Fast feedback from frequent test activities makes the developers more confident in making changes, as they are instantly informed if something goes wrong.

Continuous delivery builds upon the idea of continuous integration. The term was popularized by Humble and Farley (2010) who state that in continuous delivery "every change is, in effect, a release candidate". This means that every revision of the baseline should be evaluated through an integration pipeline, to be potentially deployed to the customers. As found by Rodriguez (2016), many sources treat continuous delivery and continuous deployment as one and the same. However, the most common interpretation is that continuous delivery is about ensuring the software can be released and deployed to production at any time (Chen 2015a, Soni 2015, Chen 2015b, Krusche and Alperowitz 2014), but may not actually be thus released and/or deployed. Continuous deployment, on the other hand, is to actually put those candidates into production (Zhu et al. 2016) with as much automation as possible (Shahin 2015).

Humble and Farley (2010) describe the principal benefit of continuous delivery as that "it creates a release process that is repeatable, reliable, and predictable". Humble and Farley also describes how continuous delivery is *empowering the development teams*, *reducing errors*, and *lowering stress*. Continuous delivery is empowering the development teams, as it is allowing them to decide when to test their latest changes in the complete product, instead of waiting for a long-term integration cycle. When Humble and Farley describe that continuous delivery is reducing errors, they refer specifically to errors introduced by manual configuration management (which is replaced by automated processes and mechanisms). Continuous delivery is also reducing stress in the organization, as the organization can feel confidence in a continuously tested product instead of being nervous before an approaching release date.

1.1.3 Limitations Imposed by Large-scale and Proximity to Hardware

Cars, telecommunication systems and video surveillance systems are all examples of large-scale software systems combined with electronic and mechanical systems – *large-scale software-intensive embedded systems*. Companies in these or similar industry segments are now adapting agile ways of working, and want to be able to utilize the benefits from continuous integration and continuous delivery. Dependencies to electronic and mechanical systems increase the complexity in a software system, and the need for short iterations is at least as evident in large-scale software-intensive embedded systems as in other industry segments.

Scaling of software development projects has been an issue since the introduction of structured development models in the 1970s, but this was not in focus when continuous integration and continuous delivery were introduced. Martin Fowler's definition of

continuous integration speaks of “members of a team” which is hard to directly relate to as hundreds or even thousands of developers in a large-scale organization. As the build time for a system is directly correlated with the size of the system, a large-scale system implies problems with long build times – which will come in direct conflict with the frequent integrations generated by the developers in a large-scale organization (Larman and Vodde 2010). In a large-scale organization it is also no longer possible to stick to “every change is a release candidate” in continuous delivery, as this would demand access to test resources that no organization can afford. Even when every change is not treated as a separate release candidate, a large-scale system still implies more, longer and more resource-consuming tests.

Proximity to hardware (electronic and mechanical systems) introduces several challenges and limitations. A test environment with real hardware is often a scarce and valued resource (Engblom 2015). Limited availability of test environments then becomes a problem, as it implies long feedback loops for the developers. If the product is a vehicle such as a car or an aircraft, this also adds new challenges (Knauss et al. 2015b). Some of the testing will then be manual testing of that e.g. the vehicle responds in a way that feels right when the steering wheel or the joystick is pushed just so. Manual test activities are challenging to combine with continuous integration and delivery, as they are hard to execute as often as requested. Some software-intensive embedded systems are hard real-time systems, with scheduled communication both within and between computers. This implies a need for coordination of software changes due to the tightly coupled architecture in the software system. This will then slow down the frequency of integration of the software, which implies longer feedback loops for the developers.

Companies that develop large-scale software-intensive embedded systems can also utilize the benefits from continuous integration and continuous delivery, but only if continuous integration and continuous delivery could be adapted to challenges and limitations introduced by large-scale and by proximity to hardware. This leads us to the topic of this thesis: *continuous integration and delivery applied to large-scale software-intensive embedded systems.*

1.2 Related Work

Previous research discuss a wide range of “difficulties” or “challenges” related to continuous integration combined with large-scale systems or embedded systems. With a few exceptions, existing publications describe one or a few problem areas, leaving out areas that others consider to be the core issues. Only a few papers describe a range of limitations or challenges. Sekitoleko et al. (2014) describe the challenges associated with technical dependencies between teams in a large-scale agile software development as planning, task prioritization, knowledge sharing, code quality and integration (handling of merge conflicts). Owen Rogers (2004) discusses problems related to large-scale continuous integration in terms of “more people” and “more code”, and proposes five strategies for successfully scaling continuous integration: establish a maximum build length, create targeted builds, write faster unit tests, smaller teams with local integration servers, and modularize the code base. However, existing publications tend

to leave out areas that other authors consider to be the core issues – no one summarizing all challenges that must be taken into account.

As described in Section 1.1, continuous integration and continuous delivery were not developed primarily for large-scale organizations. In addition to this, some divergence or even confusion exists related to the exact definition of the terms, especially related to the terms continuous delivery and continuous deployment (Rodriguez 2016). Alternative interpretations also exist for continuous integration, e.g. Fitzgerald and Stol (2015) presents continuous integration as a superset of "continuous activities" which includes continuous deployment and continuous delivery. Due to this, there is a need for clear and viable definitions of continuous integration and continuous delivery, applicable also for large-scale organizations.

In a large-scale implementation of continuous integration and continuous delivery, test activities are assembled to a pipeline which splits the test process into multiple stages. This pipeline has been referred to with different terminology, e.g. as "integration pipeline" or "deployment pipeline" (Humble and Farley 2010), "continuous integration pipeline" (Zampetti et al. 2017), "continuous delivery pipeline" (Gmeiner et al. 2015, Wettinger et al. 2017) or "continuous integration and delivery pipeline" (Vassallo et al. 2016). Many publications propose ideas or solutions that could help practitioners to improve their implementations of continuous integration and delivery, but often focusing on one particular topic, e.g. prioritization of unit tests. However, practitioners need guidelines for which continuous integration impediments their organization should focus on in order to enable more frequent integration of software. Is e.g. prioritization of unit tests important right now, or are other areas more important to improve?

A few publications discuss or touch upon how the continuous integration and delivery pipeline should be designed. Vassallo et al. (2016) summarize "practices adopted within the continuous delivery pipeline", but do not discuss strengths or weaknesses of particular test techniques or test activities. Mihindukulasooriya et al. (2016) propose a continuous integration workflow with two main steps: exploratory testing and "fine-grained analysis" (based on unit tests). Rathod and Surve (2015) present a test framework for continuous integration and delivery ("Test Orchestration") which includes test activities on different levels. Marijan (2015) introduces a multi-perspective approach for regression test case prioritization based on "business perspective, performance perspective, and technical perspective", in order to prioritize test cases for different stakeholders. Most of the existing publications do not compare different types of test activities or give any guidance on when e.g. tests on system level is better than unit tests. There are a few exceptions (Liu 2014, Rathod and Surve 2015, Marijan 2015), but still without a holistic perspective which covers the whole continuous integration and delivery pipeline.

1.3 Problem Statement and Research Questions

1.3.1 Problem Statement

Based on the background described in Section 1.1 and the summary of related work in Section 1.2, the following problem statement is the starting point for this thesis:

Continuous integration and continuous delivery have evolved to be firmly established in the mainstream of the software engineering industry. The practices are now being applied to different industry segments, including companies that develop large-scale software systems combined with electronic and mechanical systems (large-scale software-intensive embedded systems). However, continuous integration and continuous delivery were originally defined for smaller settings and not for development of large-scale systems, which is causing confusion. Due to this, there is a need for clear and viable definitions of continuous integration and continuous delivery, applicable also for large-scale organizations. Furthermore, a wide range of "difficulties" or "challenges" related to continuous integration combined with large-scale systems or embedded systems are discussed in published literature, but no existing publication summarizes all challenges that must be taken into account. A holistic perspective is lacking, which covers the whole continuous integration and delivery pipeline. Practitioners need guidelines and specific methods for how to identify their impediments, and how to better implement continuous integration and continuous delivery in organizations that develop large-scale software-intensive embedded systems.

1.3.2 Research Questions

Four research questions have been derived from the problem statement (presented in Section 1.3.1). The research questions have been the drivers behind the research studies presented in this thesis. The four research questions are:

- RQ1: *What are the challenges that must be taken into account when applying continuous integration to large-scale software-intensive embedded systems?*
- RQ2: *How should continuous integration and continuous delivery be defined for development of large-scale software systems?*
- RQ3: *How can a model be defined that shows what companies should prioritize to improve their implementations of continuous integration and continuous delivery?*
- RQ4: *How should the continuous integration and delivery pipeline be designed for large-scale software-intensive embedded systems?*

1.4 Thesis Overview

1.4.1 Research Studies Presented in the Thesis

This thesis presents the results from a number of research studies. The research studies were fully or partly addressing one of the four research questions described in Section 1.3.2. Figure 1 shows how the studies are connected to one of the four research questions (shown with colors and labeled as RQ1-RQ4) and the main relations between the studies.

Sections 1.4.2-1.4.5 describe how each of the four research questions have been addressed with one or several of the research studies.

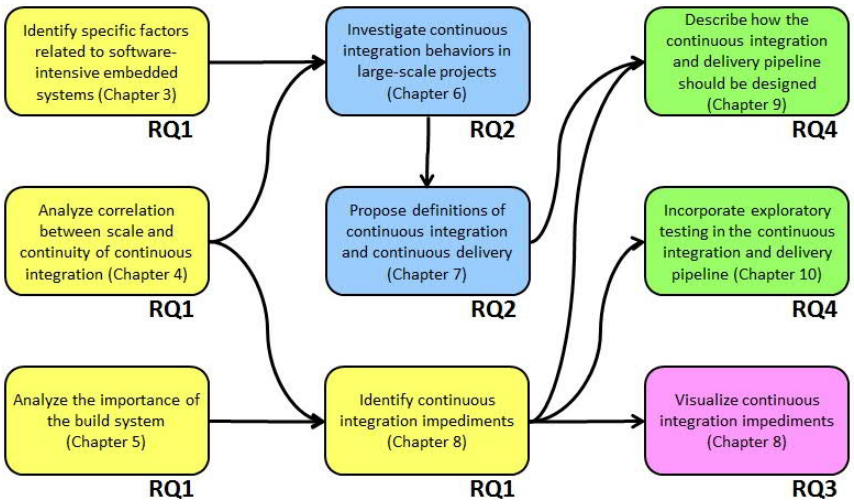

Figure 1: The research studies presented in the thesis.

1.4.2 Research Studies Addressing RQ1

Research question RQ1 ("What are the challenges that must be taken into account when applying continuous integration to large-scale software-intensive embedded systems?") was addressed in four research studies:

- *Identify specific factors related to software-intensive embedded systems* (presented in Chapter 3)
- *Analyze correlation between scale and continuity of continuous integration* (presented in Chapter 4)
- *Analyze the importance of the build system* (presented in Chapter 5)
- *Identify continuous integration impediments* (presented in Chapter 8)

The first step to address RQ1 was to *identify specific factors related to software-intensive embedded systems* which must be taken into account when applying

continuous integration to software-intensive embedded systems. The study was based on the researchers' own experiences from two industry cases, and mapped a number of factors related to software-intensive embedded systems to a list of continuous integration corner-stones found in literature.

This was followed by a study which was to *analyze correlation between scale and continuity of continuous integration*. The study was based primarily on quantitative data from six industry cases, and was validated with interviews with interviewees from five case study companies. This was complemented with another study investigating problems related to continuous integration and scale, with the objective to *analyze the importance of the build system*. The study analyzed metrics and interview results from a large-scale industry project before and after the introduction of a new build system, and identified three factors that make developers deliver software to the mainline less frequently.

Finally, based on results from the previous studies (as shown in Figure 1) a study was conducted to *identify continuous integration impediments* in large-scale industry projects developing software-intensive systems. This study included a larger group of interviewees from two case study companies, and identified twelve factors as continuous integration impediments.

1.4.3 Research Studies Addressing RQ2

Research question RQ2 ("How should continuous integration and continuous delivery be defined for development of large-scale software systems?") was addressed with two research studies:

- *Investigate continuous integration behaviors in large-scale projects* (presented in Chapter 6)
- *Propose definitions of continuous integration and continuous delivery* (presented in Chapter 7)

The study to *investigate continuous integration behaviors in large-scale projects* built on the previous studies for RQ1 (as shown in Figure 1). The study included interviews with developers from two case study companies, covering both how the developers currently committed software to the mainline and how the developers would prefer to commit their software. The result of the study was a proposed approach for how continuous integration should be interpreted for a large-scale system.

The next study's objective was to *propose definitions of continuous integration and continuous delivery* applicable for development of large-scale systems, primarily based on a systematic mapping study of published literature. The study also included an analysis of how the terms continuous integration, continuous delivery, continuous deployment and DevOps are used in literature.

1.4.4 Research Studies Addressing RQ3

Research question RQ3 ("How can a model be defined that shows what companies should prioritize to improve their implementations of continuous integration and continuous delivery?") was addressed with one research study:

- *Visualize continuous integration impediments* (presented in Chapter 8)

The study to *visualize continuous integration impediments* built on the study for research question RQ1 that was *identifying continuous integration impediments* (both studies presented in Chapter 8). To answer research question RQ3, a model was developed which allows companies to explicate a representation of the organization's current situation regarding twelve continuous integration impediments, and visualizes what the organization must focus on in order to enable more frequent integration of software. In that way, the model provides a framework that shows companies which areas they should prioritize in order to improve their implementations of continuous integration and continuous delivery.

The model was validated in workshops and interviews in two phases. The first phase of the validation included five organizations in order to validate the model in different contexts. In the second phase, the number of individuals involved in the validation was extended and different setups for the assessment workshops were compared.

1.4.5 Research Studies Addressing RQ4

Research question RQ4 ("How should the continuous integration and delivery pipeline be designed for large-scale software-intensive embedded systems?") was addressed with two research studies:

- *Describe how the continuous integration and delivery pipeline should be designed* (presented in Chapter 9)
- *Incorporate exploratory testing in the continuous integration and delivery pipeline* (presented in Chapter 10)

The studies conducted to address research question RQ4 zoomed in at some of the impediments that were identified in the study that was *identifying continuous integration impediments* (presented in Chapter 8). Both studies which were addressing research question RQ4 resulted in guidelines to be used when selecting test activities for the continuous integration and delivery pipeline for large-scale software-intensive embedded systems.

The study with the objective to *describe how the continuous integration and delivery pipeline should be designed* built on the terminology established by the studies that addressed research question RQ2 (presented in Chapters 6-7). The result from the study was a new model that describes how the continuous integration and delivery pipeline can be designed to include test activities that support four stakeholder interests. The study was based on data from four case study companies.

The second study that addressed research question RQ4 had the objective to *incorporate exploratory testing in the continuous integration and delivery pipeline.* A

new test method was developed which utilizes experienced engineers to identify complex integration problems. The test method was validated in a case study company based on interview results and quantitative data.

1.5 Personal Contribution

This thesis contains articles resulting from research and work by multiple authors. This section describes personal contributions to each of the articles.

Chapter 3: Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences (Torvald Mårtensson, Daniel Ståhl, Jan Bosch)
Research design, data gathering and analysis for this article was a joint effort between myself and Daniel Ståhl. I was performing the larger share of the work and wrote the resulting paper. Ståhl reviewed the paper, while Jan Bosch acted as mentor, sounding board and reviewer.

Chapter 4: The Continuity of Continuous Integration: Correlations and Consequences (Daniel Ståhl, Torvald Mårtensson, Jan Bosch)
Research design and analysis of the quantitative and qualitative data for this article was a joint effort by myself and Daniel Ståhl. Ståhl collected the primary case data and parts of the validation data, and wrote the paper. I collected the majority of the validation data and acted as a reviewer. Jan Bosch acted as mentor, sounding board and reviewer.

Chapter 5: Continuous Integration Is Not About Build Systems (Torvald Mårtensson, Pär Hammarström, Jan Bosch)
Research design and analysis of the quantitative and qualitative data for this article was a joint effort by myself and Pär Hammarström. My contribution to this article was ideation, collection of all the data, analysis and the writing of the paper. Hammarström reviewed the paper, while Jan Bosch acted as mentor, sounding board and reviewer.

Chapter 6: Continuous Integration Behaviors in Large-Scale Industry Projects (Torvald Mårtensson, Daniel Ståhl, Jan Bosch)
This chapter of the thesis is based on the article Continuous Integration Impediments in Large-Scale Industry Projects (described in Section 1.6). The research design for this article was a joint effort by myself and Daniel Ståhl. I collected the majority of the data, analyzed the data and wrote the paper. Ståhl collected parts of the data and acted as a reviewer. Jan Bosch acted as mentor, sounding board and reviewer.

Chapter 7: Continuous Practices and DevOps: Beyond the Buzz, What Does It All Mean? (Daniel Ståhl, Torvald Mårtensson, Jan Bosch)
Research design, validation of the search algorithm, analysis of the mapping study, and the definitions of the continuous practices in this article was a joint effort between myself and Daniel Ståhl, with Ståhl performing the larger share of the work. Jan Bosch acted as mentor, sounding board and reviewer.

Chapter 8: Enable More Frequent Integration of Software in Industry Projects (Torvald Mårtensson, Daniel Ståhl, Jan Bosch)
The research design for this article was a joint effort by myself and Daniel Ståhl. I collected the majority of the data, performed all the validation interviews and workshops, analyzed the data and wrote the paper. Ståhl collected parts of the qualitative data and acted as a reviewer. Jan Bosch acted as mentor, sounding board and reviewer.

Chapter 9: Test Activities in the Continuous Integration and Delivery Pipeline (Torvald Mårtensson, Daniel Ståhl, Jan Bosch)
The research design for this article was a joint effort by myself and Daniel Ståhl. My contribution was ideation, collection of all the data, analysis and the writing of the paper. Ståhl acted as a reviewer, while Jan Bosch acted as mentor, sounding board and reviewer.

Chapter 10: Exploratory Testing of Large-Scale Systems – Testing in the Continuous Integration and Delivery Pipeline (Torvald Mårtensson, Daniel Ståhl, Jan Bosch)
The research design for this article was a joint effort by myself and Daniel Ståhl. My contribution to this article was ideation, collection of all the data, analysis and the writing of the paper. Ståhl acted as a reviewer, while Jan Bosch acted as mentor, sounding board and reviewer.

1.6 Related Publications

Two related publications are not included in this thesis:

Mårtensson, T., Ståhl, D. and Bosch, J. (2017). Continuous Integration Impediments in Large-Scale Industry Projects. IEEE International Conference on Software Architecture, ICSA 2017, pp. 169-178. Material from this conference paper is partly included in Chapter 6, and partly in Chapter 8.

Mårtensson, T., Ståhl, D. and Bosch, J. (2017). The EMFIS model - Enable more frequent integration of software. 43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, pp. 10-17. This conference paper is a shorter presentation of parts of the work which also is described in Chapter 8.